First person action networking game using virtual reality goggles in the Unity game engine

**Praca inżynierska   
 na kierunku Informatyka**

**Pierwszoosobowa sieciowa gra akcji dla gogli wirtualnej rzeczywistości**

**w silniku Unity**

**Kraków 2019**

**Politechnika Krakowska  
im. Tadeusza Kościuszki**

Wydział Fizyki, Matematyki i Informatyki

**Dawid Lubera**

Numer albumu: 117597

Praca wykonana pod kierunkiem:  
**dr inż. arch. Paweł Ozimek**

**Uzgodniona ocena:**......................................

...........................................................................podpisy promotora i recenzenta





SPIS TREŚCI

[1. WSTĘP 4](#_Toc535178643)

[1.1. CEL PRACY 4](#_Toc535178644)

[1.2. ZAKRES PRACY 4](#_Toc535178645)

[2. WPROWADZENIE TEORETYCZNE 5](#_Toc535178646)

[2.1. KOMUNIKACJA W SIECI 5](#_Toc535178647)

[2.2. TECHNOLOGIE 5](#_Toc535178648)

[2.2.1. Unity 5](#_Toc535178649)

[2.2.2. Oculus Rift 6](#_Toc535178650)

[2.2.3. OpenVR 8](#_Toc535178651)

[2.2.4. ŚRODOWISKO PROGRAMISTYCZNE 9](#_Toc535178652)

[2.3. TECHNOLOGIE SIECIOWE W UNITY 12](#_Toc535178653)

[2.3.1. High Level API 12](#_Toc535178654)

[2.3.2. Low Level API 13](#_Toc535178655)

[2.3.3. Photon Unity Networking 13](#_Toc535178656)

[2.4. TECHNOLOGIE WIRTUALNEJ RZECZYWISTOŚCI 13](#_Toc535178657)

[2.4.1. Śledzenie pozycji 13](#_Toc535178658)

[2.4.2. Śledzenie ruchów głowy 15](#_Toc535178659)

[2.4.3. Śledzenie ruchów dłoni 15](#_Toc535178660)

[3. TWORZENIE APLIKACJI 17](#_Toc535178661)

[3.1. SPECYFIKACJA GRY 17](#_Toc535178662)

[3.2. PROJEKT 17](#_Toc535178663)

[3.2. MAPA 21](#_Toc535178664)

[3.3. KONFIGURACJA RUCHU DŁONI 24](#_Toc535178665)

[3.4. PORUSZANIE POSTACI 26](#_Toc535178666)

[3.4.1. Przemieszczanie modelu w przestrzeni 26](#_Toc535178667)

[3.4.2. Odwrotna kinematyka postaci 29](#_Toc535178668)

[3.5. KOMUNIKACJA SIECIOWA 33](#_Toc535178669)

[3.5.1. Wybór API 33](#_Toc535178670)

[3.5.2. Synchronizacja ruchu postaci 33](#_Toc535178671)

[3.5.3. Wyszukiwarka dostępnych serwerów 36](#_Toc535178672)

[3.6. STAN GRY 36](#_Toc535178673)

[3.7. DODAWANIE DŹWIĘKÓW 36](#_Toc535178674)

[4. TESTOWANIE 37](#_Toc535178675)

[5. ZAKOŃCZENIE 38](#_Toc535178676)

[5.1. PODSUMOWANIE 38](#_Toc535178677)

[5.2. MOŻLIWOŚCI ROZWOJU 38](#_Toc535178678)

[BIBLIOGRAFIA 39](#_Toc535178685)

[SPIS ILUSTRACJI 40](#_Toc535178686)

[DODATEK A. MODELE UŻYTE W PROJEKCIE 42](#_Toc535178687)

# WSTĘP

## 1.1. CEL PRACY

Celem pracy jest stworzenie gry w silniku Unity. Aplikacja powinna być stworzona z myślą o goglach wirtualnej rzeczywistości. Jednocześnie powinna używać technologii sieciowych do przesyłania położenia użytkownika. Celem gry jest wygranie w pojedynku bokserskim z drugim użytkownikiem. Obaj użytkownicy będą połączeni z serwerem i stan gry oraz położenia obu graczy w przestrzeni powinien być zsynchronizowany.

## 1.2. ZAKRES PRACY

W pracy znajduje się opis wszystkich używanych w procesie tworzenia technologii, opis implementacji poszczególnych komponentów aplikacji, sposoby testowania aplikacji, wnioski, które zostały wyciągnięte podczas pracy nad aplikacją oraz niezrealizowane pomysły, które pozwoliłyby rozwinąć aplikację.

# WPROWADZENIE TEORETYCZNE

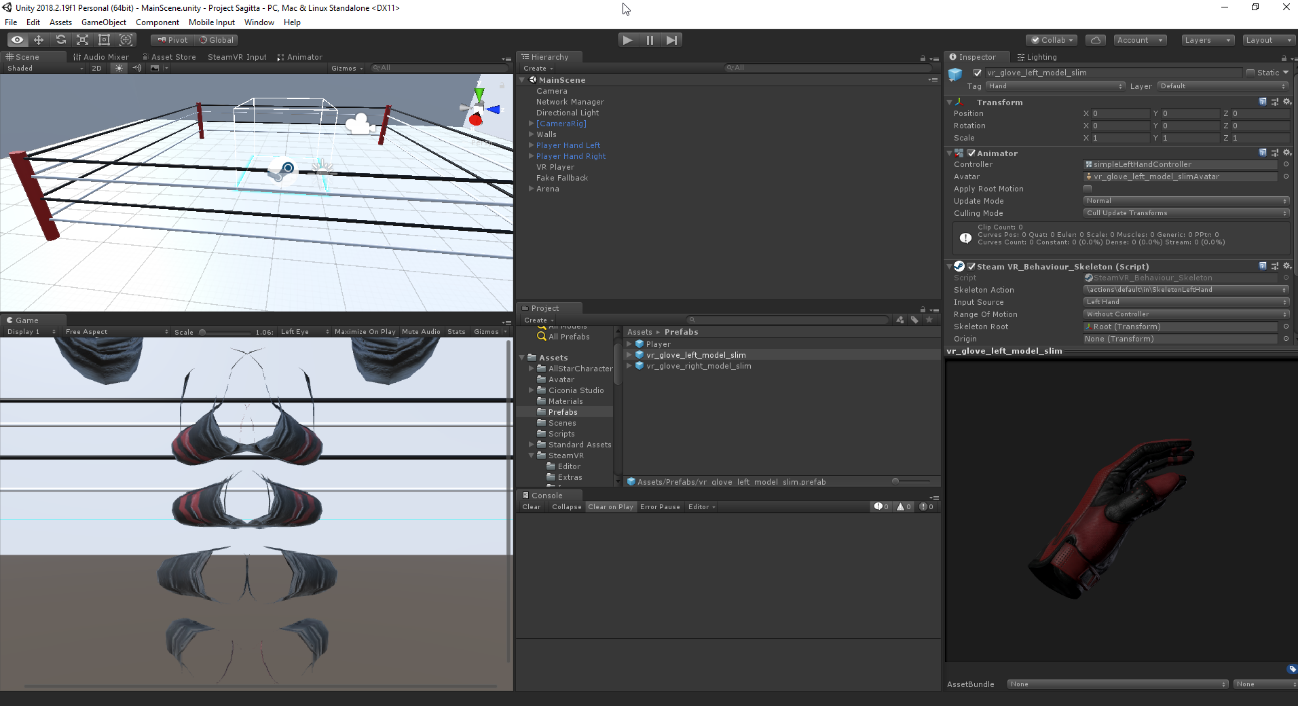
## KOMUNIKACJA W SIECI

## TECHNOLOGIE

### 2.2.1. Unity

Unity jest rozbudowanym silnikiem do gier, który pozwala na tworzenie, wyświetlanie i zarządzanie dwu i trójwymiarowymi światami. Darmowa wersja programu pozwala na korzystanie ze znacznej większości funkcji. Ograniczenia wiążą się z brakiem wsparcia producenta, ilości graczy w rozgrywkach dla wielu graczy oraz maksymalny dochód ze sprzedaży, po którym wymagana będzie płatna licencja. Środowisko to pozwala na kompilację gry na wiele różnych platform w tym: Windows, Linux, Mac OS X, PS4, Xbox One, Nintendo Switch, Android, iOS.

W 2005 roku Unity zostało stworzone jako silnik do gier trójwymiarowych i wciąż jest to jego główne zadanie. Zawiera jednak wiele funkcji pozwalających w łatwy sposób tworzyć gry dwuwymiarowe. Pozwala także tworzyć gry i aplikacje dla gogli wirtualnej rzeczywistości. W przypadku niektórych gogli wymagane jest pobranie odpowiednich zasobów ze sklepu Unity – *Unity Asset Store[[1]](#footnote-1)*.



Rysunek 1. Główne okno edytora Unity, opracowanie własne

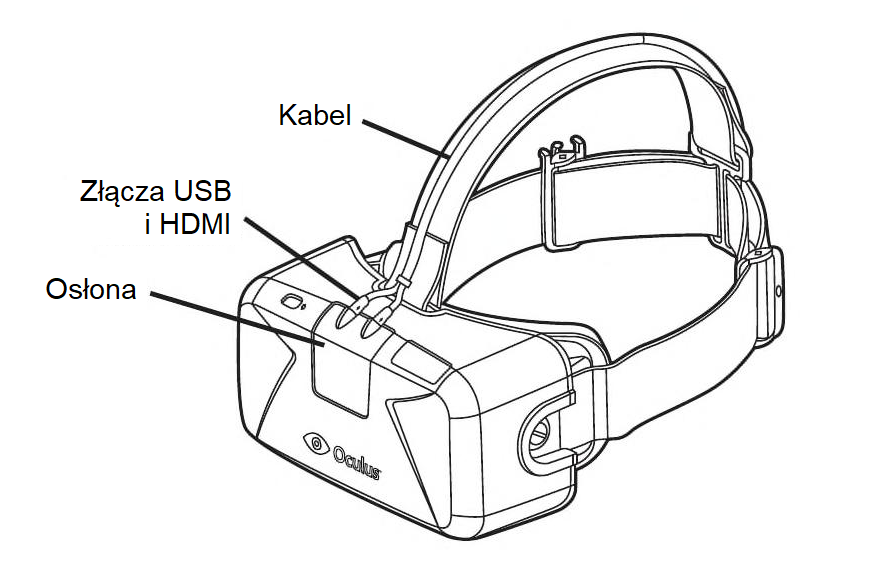
Edytor Unity pozwala na importowanie do gry modeli w wielu popularnych formatach w tym plików z programów: 3ds Max, Maya, Blender, SketchUp, a także popularny w wielu innych programach format OBJ. Umożliwia import tekstur  
i materiałów bezpośrednio z plików modeli lub osobno w postaci plików graficznym między innymi w formatach JPEG i PNG.

Pracując w Unity część zadań można wykonać bez programowania. Tworzenie świata gry, proste łączenia elementów lub animacji nie wymagają pisania kodu. Te czynności można wykonać poprzez przeciąganie odpowiednich elementów, wybór odpowiednich trybów pracy czy zaznaczanie pól wyboru. Przy programowaniu używany jest język C#. Twórcy programu starają się na bieżąco wprowadzać nowe funkcjonalności języka do swojego silnika. Unity posiada także własny sklep, w którym można kupić gotowe skrypty, modele, pliki muzyczne czy narzędzia dodające nowe funkcjonalności do edytora.

### 2.2.2. Oculus Rift

Oculus Rift są pierwszymi goglami wirtualnej rzeczywistości, które odniosły komercyjny sukces. Pierwsza deweloperska została wydana w marcu 2013 roku, a wersja konsumencka w styczniu 2016 roku. W grudniu tego samego roku zostały wydane kontrolery Touch.

Gogle wirtualnej rzeczywistości pozwalają na obserwowanie wirtualnego świata w sposób symulujący obecność w tym świecie. Wyświetlają one dwa obrazy, po jednym dla każdego oka, które zasłaniają całe pole widzenia. Stosowane są odpowiednio zakrzywione soczewki oraz ruch kamery w grze lub filmie bazujący na faktycznych ruchach głowy. Sprawia to, że ludzki mózg faktycznie jest w stanie wczuć się w wydarzenia, które odbywają się na ekranach.



Rysunek 2. Schemat Oculus Rifta[[2]](#footnote-2)

Kontrolery Touch posiadają dokładne sensory pozwalające analizować pozycję, ruch i obrót dłoni użytkownika, a dokładne przyciski dające informację o sile nacisku. Kontrolery pozwalają na dokładne symulowanie dłoni i odwzorowanie ruchu każdego palca. Sprawienie, że gracz widzi ruchy swoich dłoni w grze sprawia, że immersja[[3]](#footnote-3) jest jeszcze głębsza. Na każdym z kontrolerów znajdują się dwa zwykłe przyciski pozwalające odczytywać stany czy jest wciśnięty – true czy nie – false, niewielki joystick kierunkowy, dotykowe miejsce obok przycisków, a także dwa triggery. Triggery w przeciwieństwie do przycisków pozwalają odczytać nie tylko informacje czy przycisk jest wciśnięty, czy nie, ale także siłę nacisku – wartość od 0 do 1. Jeden trigger jest umieszczony w pozycji palca wskazującego, a drugi, trzech palców: środkowego, serdecznego i małego. Każdy element zwraca także informację, czy użytkownik go dotyka, czy nie. Te wszystkie dane pozwalają stwierdzić w, jaki sposób jest ułożona dłoń użytkownika.



Rysunek 3. Kontrolery Oculus Touch[[4]](#footnote-4)

### 2.2.3. OpenVR

OpenVR jest zestawem deweloperskim (SDK) stworzonym przez firmę Valve, aby ujednolicić tworzenie oprogramowania dla urządzeń wirtualnej rzeczywistości. Valve miało kluczowe znaczenie przy tworzeniu Oculus Rifta, a także stworzyło gogle HTC Vive i kontrolery dla dłoni Valve Knuckles. Wszystkie te urządzenia są wspierane przez OpenVR, ale lista urządzeń, z których można używać z tym SDK jest znacznie dłuższa. Valve jest też twórcą dodatku do Unity dodającego wsparcie dla tego interfejsu.

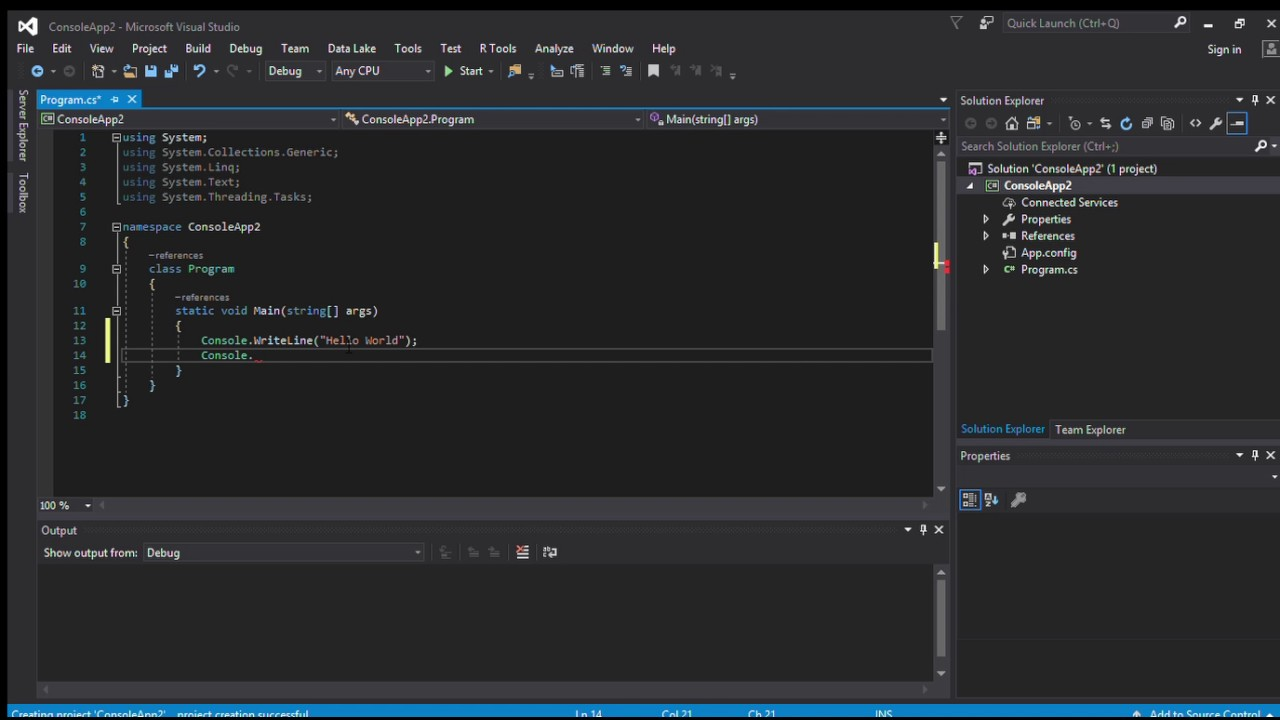


Rysunek 4. Okno przypisywania przycisków do akcji, opracowanie własne

Gry stworzone przy pomocy OpenVR pozwalają także na bardzo łatwą zmianę zastosowań danych przycisków zarówno przez deweloperów, jak i przez graczy. Pozwala także na analizę użycia przycisków w trakcie testowania aplikacji.

### 2.2.4. ŚRODOWISKO PROGRAMISTYCZNE

Edytorem kodu, który został użyty w projekcie było Visual Studio. Głównym powodem takiego wyboru jest fakt, że jest ono automatycznie instalowany i najbardziej wspierany przez Unity środowiskiem programistycznym na systemy Windows. Visual Studio jest zintegrowanym środowiskiem programistycznym stworzonym przez firmę Microsoft. Środowisko programistyczne pozwala na łatwiejsze zarządzanie kodem całego projektu. Visual Studio wspiera programowanie w wielu popularnych językach programowania i kodowania między innymi: C, C++, C#, JavaScript, CSS czy HTML. Twórcy Visual Studio wprowadzają obsługę nowych funkcjonalności języków w nowych aktualizacjach.

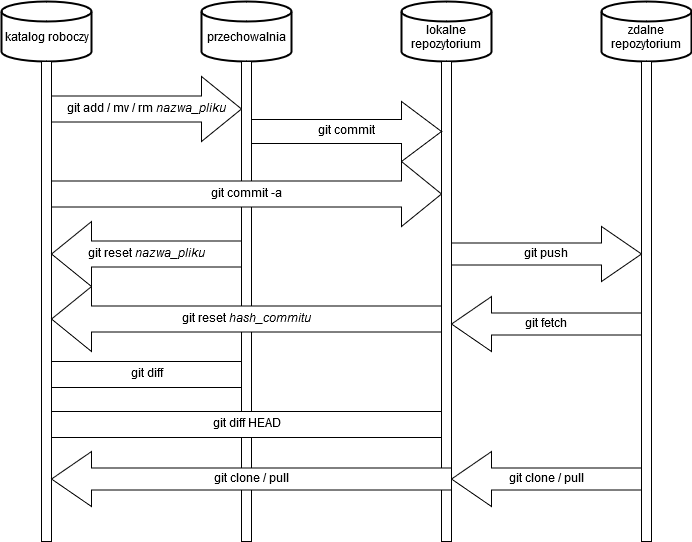
****

**Rysunek 5. Okno główne Visual Studio 2018, opracowanie własne**

Aktualnie jedynym językiem używanym przy programowaniu aplikacji w Unity jest C#. Jest to obiektowy język programowania stworzonym przez firmę Microsoft, którego pierwsza wersja ukazała się w lipcu 2000. Od tamtego czasu jest aktywnie rozwijany i wprowadzane są nowe funkcjonalności. Głównymi językami, na których bazuje C# są C++, Java i Delphi. Kod programu napisanego w tym języku jest kompilowany do specjalnego języka Common Intermediate Language. Taki program może zostać uruchomiony przez środowisko uruchomieniowe np. *.NET Framework*, *.NET Core* lub *Mono*.

System kontroli wersji jest narzędziem do śledzenia zmian w kodzie źródłowym. Narzędziem używanym w tym projekcie był Git. Przechowuje on kod źródłowy projektu w repozytorium. Podstawowa praca z systemem Git opiera się na używaniu kilku komend:

* *git add / git mv / git rm* – komendy pozwalające na dodawanie, przesuwanie i usuwanie plików w repozytorium
* *git commit* – zatwierdza dokonane zmiany i aktualizuje lokalne repozytorium
* *git push* – wysyła lokalne zmiany do *origin* – repozytorium na serwerze
* *git fetch* – pobiera z *origin* aktualną wersję projektu i umieszcza w lokalnym repozytorium, nie modyfikuje plików projektu
* *git merge* – pozwala na scalenie plików, które były modyfikowane np. przez różne osoby w tym samym momencie
* *git pull* – pobiera najnowszą wersję projektu z *origin* i umieszcza w repozytorium jednocześnie scalając pliki
* *git reset* – w zależności od podanej wartości pozwala na usunięcie zmian z pliku lub pozwala na zresetowanie aktualnego repozytorium do konkretnego commita
* *git clone* – pozwala na pobranie najnowszej wersji projektu z *origin* i stworzenie lokalnego repozytorium

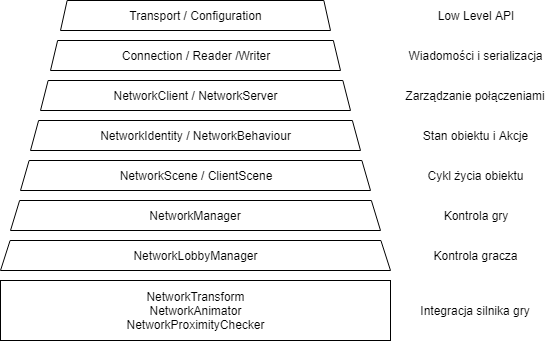


Rysunek 6. Schemat działania Git, opracowanie własne

## TECHNOLOGIE SIECIOWE W UNITY

Unity posiada dwa wbudowane interfejsy sieciowe, z czego pierwszy z nich jest bazą dla drugiego:

* Low Level API (LLAPI)
* High Level API (HLAPI)



Rysunek 7. Schemat warstwy sieciowej w Unity, opracowanie własne

### 2.3.1. High Level API

High Level API udostępnia logikę wymaganą w grach sieciowych bez konieczności wchodzenia w szczegóły przesyłania danych w protokołach sieciowych. HLAPI pozwala także na łatwe włączenie matchmakingu, czyli automatycznego parowania dwóch graczy, jeśli tylko są podłączeni do tego samego serwera. Darmowa wersja Unity pozwala na połączenie się do serwerów firmy Unity maksymalnie 20 graczy. W większości wypadków jest to bardzo wygodne i znacznie szybsze rozwiązanie. Ma ono jednak swoje wady, między innymi to, że wymusza, w jaki sposób klient komunikuje się z serwerem oraz jak wiele danych i w jakim formacie przesyła klient.

### 2.3.2. Low Level API

Low Level API zostało zaprojektowane w taki sposób, aby zapewnić znacznie większą elastyczność. Kosztem elastyczności tego rozwiązania jest jednak ilość pracy, jaką należy wykonać do stworzenia systemu pozwalającego na połączenie i rozgrywkę dwóch klientów. LLAPI jest tylko nieco bardziej przystosowaną do obsługi gier nakładką na narzędzia działające w transportowej warstwie modelu TCP/IP.

### 2.3.3. Photon Unity Networking

Photon Unity Networking (PUN) jest dodatkowym assetem[[5]](#footnote-5) do Unity. Zostało zaprojektowane w podobny sposób co HLAPI. Podstawowa konfiguracja w projekcie wymaga jednak więcej pracy z uwagi na brak gotowego rozwiązania do interfejsu  
i łączenia użytkowników. Darmowa wersja pozwala na równoległą obsługę 20 użytkowników oraz ma ograniczenie na 500 wysłanych wiadomości na sekundę.

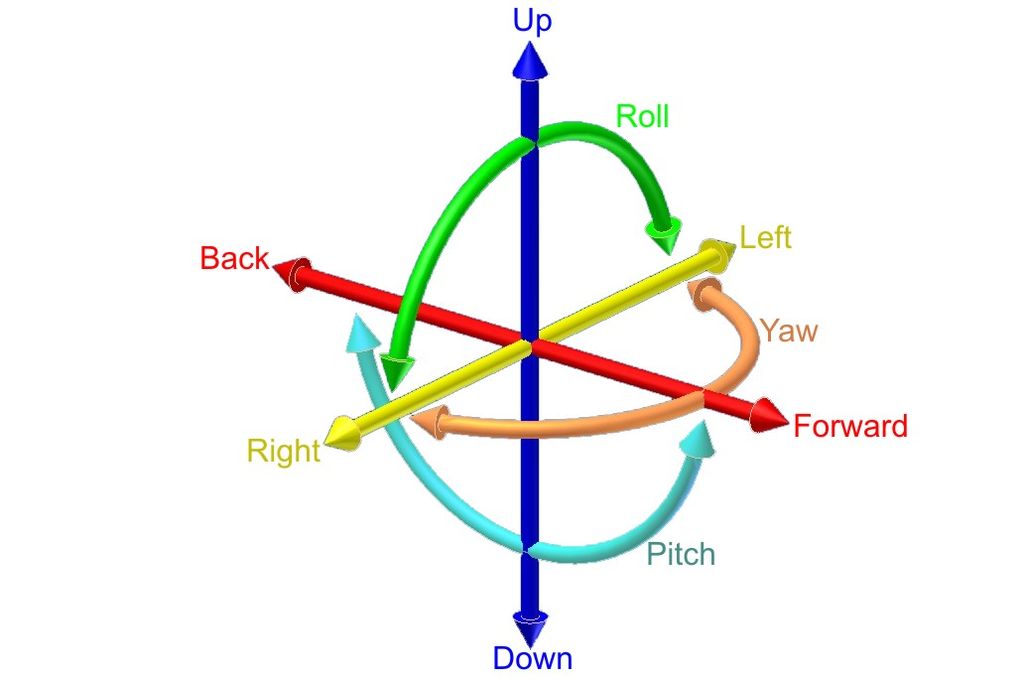
## TECHNOLOGIE WIRTUALNEJ RZECZYWISTOŚCI

### 2.4.1. Śledzenie pozycji

Śledzenie pozycji jest technologią, która pozwala urządzeniu na określenie swojej pozycji relatywnie od otoczenia. Jest to możliwe dzięki połączniu odpowiedniego sprzętu z oprogramowaniem.

Śledzenie pozycji w wirtualnej rzeczywistości przynosi wnosi wiele do samego doświadczenia. Pozwala na zmianę pola widzenia użytkownika w zależności od tego, czy porusza się, skacze, kuca czy pochyla do przodu. Z perspektywy użytkownika zwiększa połączenie świata fizycznego i wirtualnego. Pozwala także na użycie paralaksy[[6]](#footnote-6), która pomaga poinformować mózg o tym jak daleko w relacji do innych przedmiotów coś się znajduje.

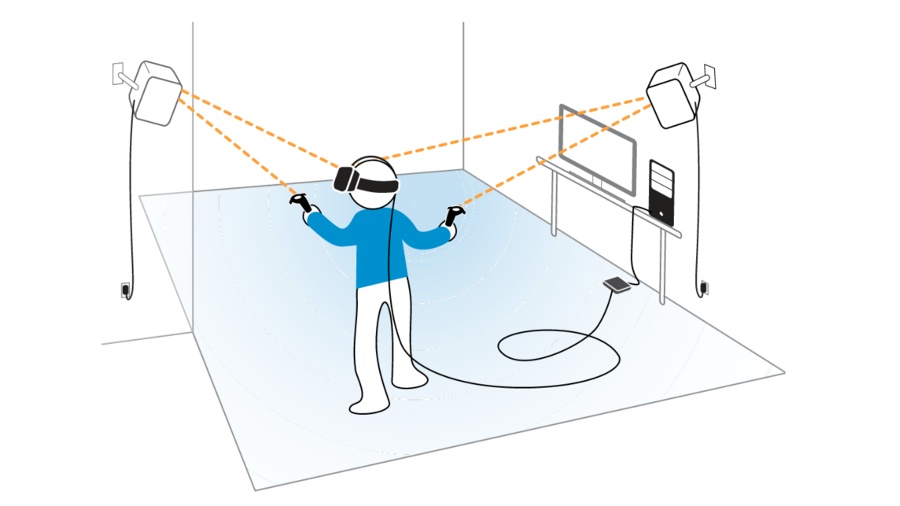
Warto zaznaczyć, że śledzenie pozycji nie jest tym samym co śledzenie ruchów głowy. To drugie pozwala tylko na śledzenie obrotów głowy, ale nie jej pozycji, ale obie technologie są wymagane, do zapewnienia tzw. sześciu stopni swobody (ang. *Six degrees of freedom, 6DoF*). Sześć stopni swobody reprezentują osie X, Y, Z oraz obroty wokół tych osi (ang. *pitch, yaw, roll*). Sześć stopni swobody pomogło znacznie zredukować występowanie efektów choroby lokomocyjnej wśród użytkowników.



Rysunek 8. Sześć stopni swobody[[7]](#footnote-7)

Istnieje kilka różnych metod śledzenia pozycji. Wybranie odpowiedniej lub wystarczającej wymagało odpowiedzi na pytania jakiej dokładności i szybkości odświeżania, powierzchni użytkowej, mocy obliczeniowej potrzebuje urządzenie, czy będzie używane z komputerem, czy samodzielnie i w końcu znalezienie konsensusu między ceną a jakością.

Oculus Rift używa dwóch sensorów, które muszą być oddalone od siebie o co najmniej jeden metr. Dla wygody użytkownika wymagane jest, aby komputer pozwalał na renderowanie obu obrazów w 90 klatkach na sekundę.



Rysunek 9. Przykład śledzenia pozycji dla VR[[8]](#footnote-8)

### 2.4.2. Śledzenie ruchów głowy

Do śledzenia ruchów głowy używa się wieloosiowych żyroskopów. Rozwój smartfonów sprawił jednak, że ta technologia jest bardzo tania i wydajna. Wiele modeli gogli korzysta ze światła podczerwonego do określania pozycji użytkownika. Tańsze gogle, które używają ekranu telefonu do wyświetlania obrazu nie obsługują tej technologii, co sprawia, że śledzenie ruchów ogranicza się właśnie do ruchu głowy. Droższe modele posiadają kilka żyroskopów, aby dokładniej liczyć rotację gogli.

Odczytywanie ruchu głowy jest drugą z kluczowych części przy sześciu stopniach swobody. Ruch ten jest zaimplementowany w podobny sposób do zmiany pozycji w wirtualnym świecie.

### 2.4.3. Śledzenie ruchów dłoni

Podobnie jak w przypadku śledzenia pozycji, śledzenie ruchów dłoni opisuje proces ciągłego przechwytywania pozycji dłoni użytkownika. Odczytane informacje pozwalają na poruszanie rękami i ciałem użytkownika w wirtualnym świecie. Używanie dłoni do interakcji z wirtualnym światem jest bardzo naturalną czynnością. Nawet użytkownicy gogli niewspierających śledzenia ruchów dłoni próbują chwytać przedmioty, mimo że bez nich nie są w stanie tego dokonać. Kontrolery pozwalają na wprowadzenie dodatkowego realizmu do wirtualnego świata. Zwykle implementacja takiego rozwiązania opiera się na stworzeniu w wirtualnym świecie dłoni, które odwzorowują ruchy użytkownika.

Istnieją dwa główne sposoby implementacji ruchów dłoni:

* Pierwsza technika wymaga posiadania odpowiednich kontrolerów, po jednym dla każdej z dłoni, na których są przyciski, triggery czy touchapdy, tę technikę wykorzystuje większość z popularnych gogli, gdyż jest ona obecnie najskuteczniejsza
* Druga technika uwzględnia zaawansowane algorytmy analizy obrazu z kilku kamer, umieszczonych zarówno na goglach, jak i w dodatkowych sensorach. Metoda ta jest jednak bardzo trudna do implementacji w wydajny i jednocześnie dokładny sposób, przez co obecnie jest rzadko wykorzystywana. Może się to jednak zmienić wraz z rozwojem technologii

Oculus Rift wykorzystuje pierwszą z tych technik w swoich kontrolerach Oculus Touch. Z tego powodu właśnie ta pierwsza technika została zaprezentowana w projekcie.

# TWORZENIE APLIKACJI

## 3.1. SPECYFIKACJA GRY

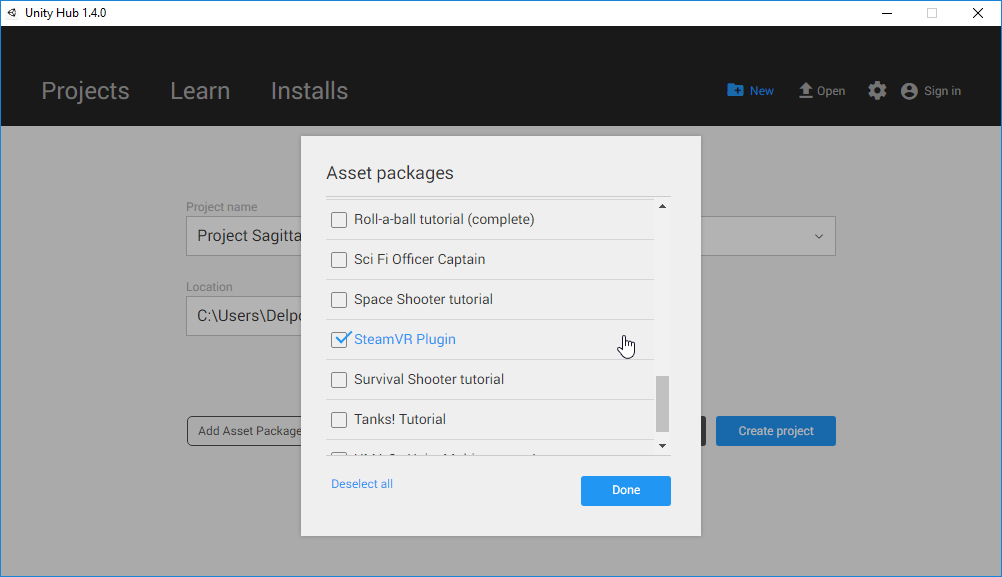
Gra pozwala na połączenie się gracza do serwera, do rozgrywki wymagane są jednak dwie osoby. W momencie gdy połączonych jest dwóch graczy, następuje faza rozgrywki w której celem każdego z użytkowników jest uderzenie głowy drugiego gracza. Uderzenie w głowę skutkuje zakończeniem rozgrywki i pojawieniem się napisu informującego o wygranej. U drugiego użytkownika pojawia się napis informujący o przegranej.

W katalogach które należą do struktury projektu znajdują się następujące obiekty:

* AllStarCharacterLibrary – model oraz dodatkowe animacje, tekstury i skrypty pobrane z assetem postaci do gry
* Materials – materiały stworzone dla projektu
* Prefabs – prefaby stworzone dla projektu
* Scenes – scena która jest używana do rozgrywki
* Scripts – skrypty, m.in. do poruszania postacią
* Standard Assets – podstawowe modele, animacje, materiały, które zostały dodane automatycznie przez Unity
* SteamVR, SteamVR\_Input – wszystkie skrypty biblioteki OpenVR

## 3.2. PROJEKT

Unity pozwala na łatwe stworzenie projektu klikając na przycisk *New* znajdujący się w prawym górnym rogu ekranu aplikacji startowej. W kolejnym kroku należy podać nazwę projektu, miejsce zapisu projektu w systemie plików oraz używany w projekcie szablon – 2D lub 3D. Okno to pozwala także na automatyczne dodanie dodatkowych assetów pobranych z *Unity Asset Store*.



Rysunek 11. Ekran tworzenia projektu w Unity, opracowanie własne

Projekt został skonfigurowany z następującymi opcjami:

* Nazwa projektu – Project Sagitta
* Szablon – 3D
* Lokalizacja – *ścieżka w systemie plików*
* Dodatkowe zasoby – SteamVR Plugin

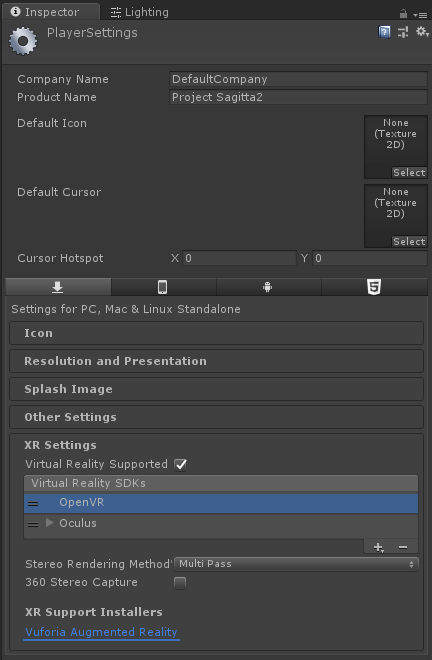
Po stworzeniu projektu przez Unity plugin SteamVR wyświetlił okno z zalecanymi ustawieniami konfiguracyjnymi projektu. Opcje, które powinny zostać zmienione to:

* Wyświetlanie okna rozdzielczości – zmiana z wyświetlania na ukrycie. W przypadku aplikacji stworzonych na VR wyświetlanie powinno być unikane, ponieważ obraz wyświetla się na ekranach w hełmie i wymaga odpowiedniej rozdzielczości.
* Umożliwienie zmiany rozmiaru okna podglądu – nie ma znaczenia dla wyświetlania gry na goglach
* Przestrzeń kolorów – zmiana z Gamma na Linear z uwagi na mniejsze obciążenie karty graficznej



Rysunek 12. Okno konfiguracji opcji pluginu SteamVR, opracowanie własne

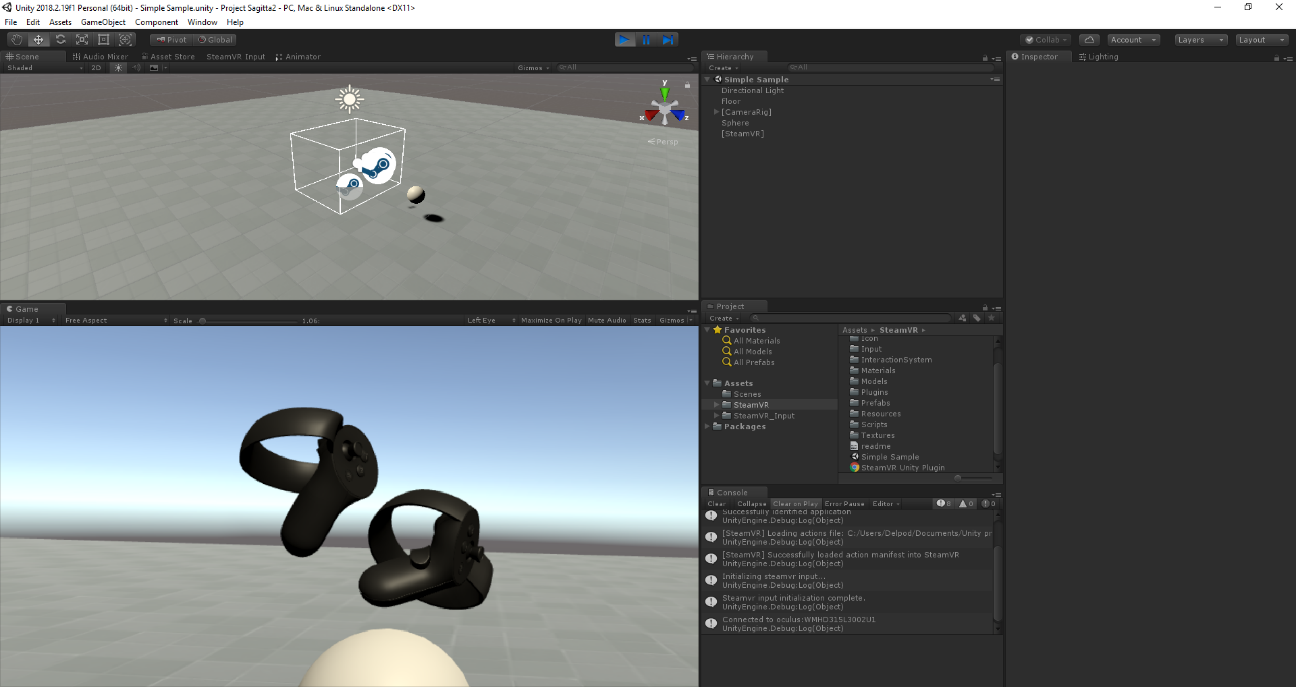
Kolejnym wymaganym krokiem jest zmiana ustawień projektu w Unity. Zostało to dokonane poprzez kliknięcie w oknie programu *Edit* > *Project Settings* > *Player*. W wyświetlanym oknie w sekcji XR zaznaczona została opcja *Virtual Reality Supported* oraz z uwagi na korzystanie z OpenVR zmieniona została kolejność SDK, tak, aby OpenVR znajdował się na pierwszym miejscu.



Rysunek 13. Konfiguracji Opcji XR, opracowanie własne

Do uruchomienia aplikacji wymagana była jeszcze scena, która zawiera kamerę oraz 2 obiekty odpowiadające każdej z dłoni. Plugin SteamVR zawiera taką podstawową scenę, więc została ona skopiowana do projektu i użyta jako baza do dalszej pracy.

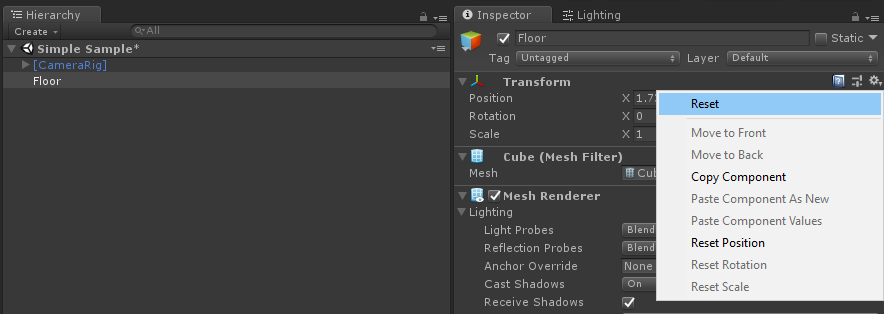
Uruchamiając projekt po raz pierwszy plugin SteamVR zaoferował możliwość konfiguracji wejścia. Podstawowe domyślne akcje zaproponowane przez plugin były wystarczające i wymagane było tylko ich wygenerowanie. Po ich zapisaniu można było już uruchomić projekt, który już na goglach wyświetlał podstawową scenę oraz umożliwiał poruszanie kontrolerami.



Rysunek 14. Okno edytora z uruchomioną podstawową sceną, opracowanie własne

## 3.2. MAPA

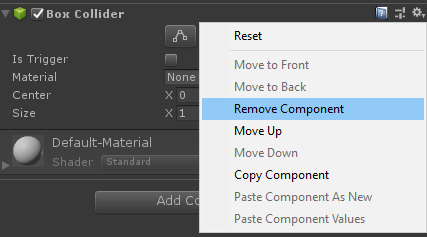
Mapa miała składać się z oświetlonego pokoju z ringiem na środku. Na początku usunięta została zawartość sceny poza obiektem *[CameraRig]***.** Klikając prawym klawiszem myszy i wybierając *3D Object* > *Cube* w oknie hierarchii utworzony został nowy prostopadłościan. Następnie należało zresetować transformację tego obiektu klikając w ikonę zębatki komponentu *Transform* i wybierając *Reset*.



Rysunek 15. Resetowanie komponentu w Edytorze, opracowanie własne

Odpowiednio ustawiając zmienne w komponencie *Transform* utworzona została podłoga o szerokości i długości 20 metrów. Obiekt stanowi podłogę pokoju. Aby utworzyć kolejne ściany pokoju podłoga została zduplikowana pięciokrotnie poprzez zaznaczenie obiektu podłogi i kliknięcie na klawiaturze *Ctrl + D* pięć razy. Zduplikowanym obiektom nadane zostały odpowiednie transformacje, które sprawiały, że wszystkie 6 obiektów stworzyło prostopadłościan wyglądający jak pokój.

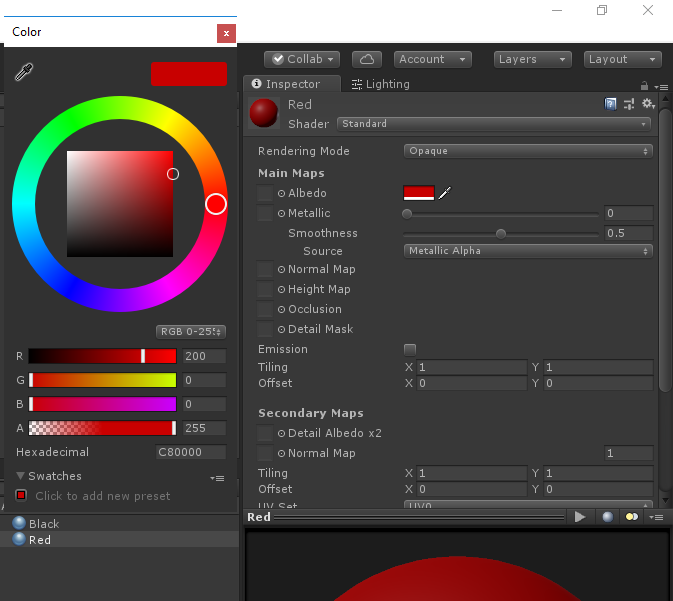
W odróżnieniu od podłogi ściany pokoju nie potrzebują wykrywania kolizji – użytkownik poruszać się będzie w obszarze mniejszym niż ściany. Ze wszystkich obiektów poza *Floor* usunięty został komponent *Box Collider*. W menu które otwiera się po otworzeniu zębatki wybrana została opcja *Remove Component.*



Rysunek 16. Usuwanie komponentu Box Collider, opracowanie własne

Następnie w podobny sposób jak ściany, klikając prawym klawiszem myszy i wybierając opcję *Create empty* utworzony pusty obiekt, który został nazwany *Walls*. Obiekt ten także został zresetowany. Wszystkie utworzone ściany zostały przeniesione w hierarchii pod obiekt *Walls* stając się jego częścią.

Kolejną częścią było utworzenie ringu. Aby stworzyć słupki na rogach najpierw w menu otwierającym się po kliknięciu prawym klawiszem myszy należało wybrać opcję *Object 3D > Cyllinder.* Po utworzeniu został on nazwany *Pole1,* transformacja obiektu została zresetowana, a komponent *Capsule Collider* został z niego usunięty. Chcąc zmienić kolor obiektu konieczne było stworzenie nowego materiału. Materiał został utworzony klikając prawym klawiszem myszy w oknie *Project* i wybierając opcję *Create > Material*. Jedyną rzeczą, która musiała zostać zmieniona w nowym materiale jest kolor. Po kliknięciu w opcję *Albedo* otwiera się okno, które umożliwia zmianę koloru.



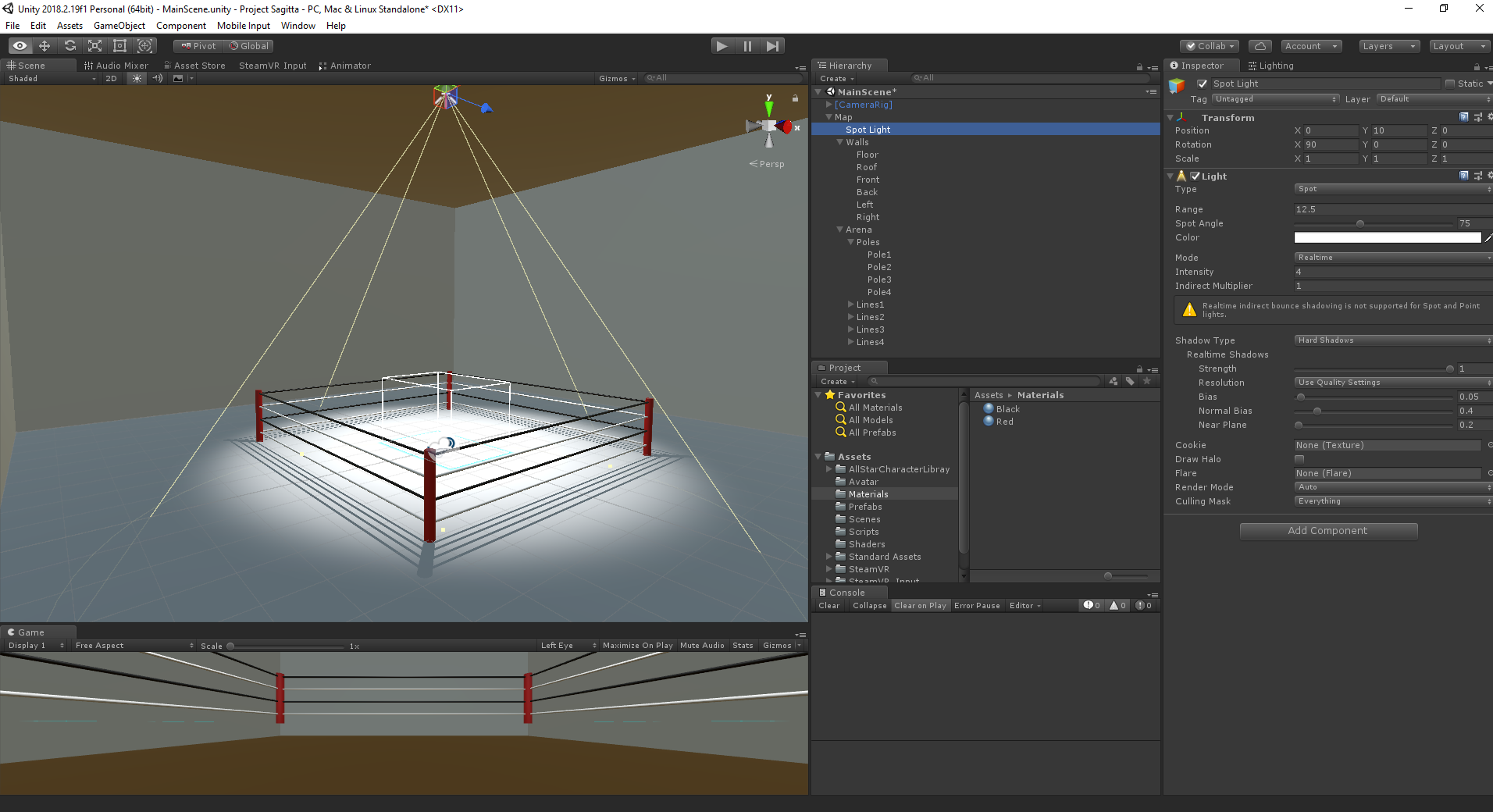
Rysunek 17. Tworzenie nowego materiału, opracowanie własne

Aby nadać kolor słupkowi należało przenieść materiał z okna *Project* na obiekt w scenie. Obiekt został trzykrotnie zduplikowany a poszczególnym słupki zostały ustawione w na planie. Podobnie jak ściany obiekty zostały zgrupowane w nowym obiekcie o nazwie *Poles.*

Następnym krokiem było utworzenie lin, które znajdują się pomiędzy słupkami. Ponownie użyte tutaj zostały cylindrów. Cztery utworzone liny zostały zresetowane, a wartości ich transformacji zostały ustawione tak, by wszystkie cztery znajdowały się między pierwszą parą słupków.

Stworzony został nowy, czarny materiał. Został on nadany obiektom *Line2*i *Line4.* Obiekty zostały zgrupowane do nowego obiektu o nazwie *Lines1.* Obiekt *Lines1* został następnie powielony trzykrotnie, a poszczególnym obiektom nadane zostały wartości transformacji, ustawiając je pomiędzy kolejnymi parami słupłów. Obiekty *Poles, Lines1, Lines2, Lines3* i *Lines4* zostały zgrupowane w obiekcie *Map.*

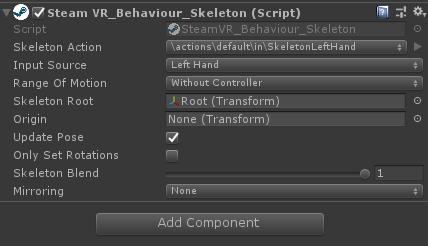
Następnie pod obiektem *Map* zostało utworzone światło typu *Spotlight,* obiekt został umieszczony na górze mapy, aby światło padało na arenę oświetlając ją, resztę pokoju pozostawiając w półmroku.



Rysunek 18. Wygląd utworzonej mapy wraz z pokazaną hierarchią obiektów, opracowanie własne

## 3.3. KONFIGURACJA RUCHU DŁONI

Do projektu zostały dodane modele dłoni, które znajdują się w folderze z plikami pluginu SteamVR. Nazwa modeli dłoni, które zostały użyte w projekcie to *vr\_glove\_left\_model\_slim* oraz *vr\_glove\_right\_model\_slim*. Do modeli dodany został komponent *Steam VR\_Behaviour\_Skeleton*. Wartości konfiguracyjne ustawione dla lewej dłoni w komponencie przedstawione są na Rys. 19. Wartości dla prawej dłoni są bardzo podobne, różnice polegają w opcji *Skeleton Action* i *Input Source*. Obie te opcje wskazują na wartości *RightHand*.



Rysunek 19. Konfiguracja komponentu szkieletu dla lewej dłoni, opracowanie własne

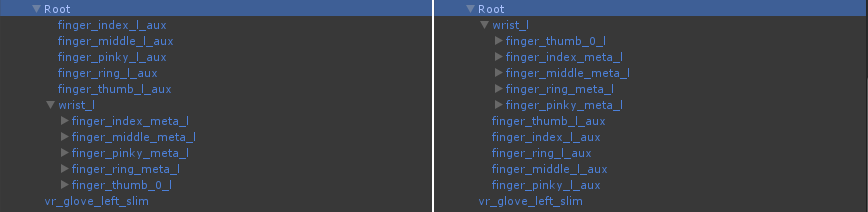
Wartości które można skonfigurować to:

* *Skeleton Action* – wybranie której akcji ma używać skrypt
* *Input Source* – z którego fizycznego kontrolera ma pobierać wartości skrypt
* *Range Of Motion* – zakres ruchów modelu, porównanie dostępnych opcji znajduje się na Rys. 20. Opcja *With Controller* jest bliższe aktualnemu układowi dłoni użytkownika, jednak nie pozwala na zamknięcie dłoni w grze, przez co odczucie immersji jest mniejsze
* *Skeleton Root* – nadrzędny obiekt dla całego systemu szkieletowego dłoni
* *Origin* – pozwala na określenie obiektu relatywnie do którego powinny poruszać się dłonie, przykładowym zastosowaniem jest np. poruszający się po mapie pojazd
* *Update Pose* – określa czy aktualizację pozycji i rotacji zajmować ma się ten skrypt czy będzie odpowiedzialny inny
* *Only Set Rotations* – opcja, która pozwala zmieniać samą rotację dłoni bez zmiany pozycji; działa tylko w połączeniu z *Update Pose*
* *Skeleton Blend* – oznacza szybkość zmiany pozycji dłoni – 1 to najszybciej, a 0 to brak zmiany pozycji
* *Mirroring* – pozwala na ustawienie lustrzane kopiowanie pozycji i rotacji kości dłoni



Rysunek 20. Efekty wyboru opcji Range of Motion: Without Controller (po lewej) i With Controller (po prawej), opracowanie własne

W dodanych modelach należało także zmienić kolejność punktów transformacji modelu. W oryginalnych rękawicach kości znajdują się w nieodpowiedniej kolejności  
i zgięcia nie odzwierciedlają faktycznego układu dłoni.



Rysunek 21. Nieprawidłowy (po lewej) i prawidłowy (po prawej) układ kości dłoni w modelu, opracowanie własne

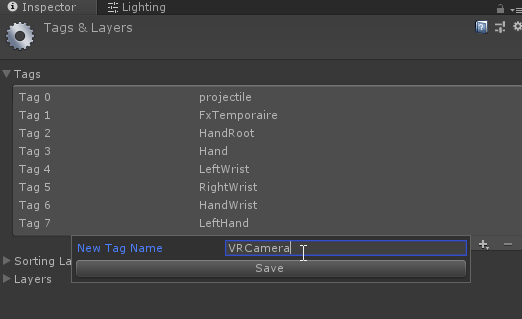
## 3.4. PORUSZANIE POSTACI

### 3.4.1. Przemieszczanie modelu w przestrzeni

Stworzenie odpowiednio przemieszczania w przestrzeni jest wymagane, aby w trakcie rozgrywki drugi użytkownik widział model gracza w sposób bliski temu,  
w którym miejscu i w jakiej pozycji aktualnie znajduje się użytkownik. Przy goglach wirtualnej rzeczywistości należy uwzględniać ruchy głowy oraz ruchy dłoni w przestrzeni świata.

Do projektu zaimportowany został model postaci pobrany z *Unity Asset Store[[9]](#footnote-9)*. Dla modelu utworzony został nowy skrypt o nazwie *PlayerController*. Skrypt posiada 3 zmienne publiczne, które następnie zostały ustawione w edytorze: *head, leftHand*i *rightHand.* W edytorze zmienne te reprezentowane są przez obiekty *Head, L Hand*i *R Hand,* które w strukturze znajdują się pod obiektem *ROOT* modelu. Używane są także zmienne *vrHead, vrLeftHand* i *vrRightHand.*

Przy komunikacji sieciowej, każdy z graczy będzie miał własne obiekty kontrolerów i głowy, przez co zmienne te muszą być zostać odnalezione dynamicznie. Aby to zrealizować potrzebne było utworzenie etykiet dla modeli dłoni. Należało użyć komendy *Edit > Project Settings > Tags and Layers* w sekcji *Tags* dodane zostały etykiety *LeftWrist, RightWrist* i *VRCamera* (Rys. 22).



Rysunek 22. Dodawanie etykiet do projektu, opracowanie własne

Etykiety zostały nadane obiektom *Camera* w *[CameraRig]* oraz obiektom *wrist\_l*  i *wrist\_r* w *ROOT* modeli lewej i prawej dłoni. W skrypcie *PlayerController* utworzone zostały funkcje odpowiedzialne za odszukanie obiektów z odpowiednimi etykietami.

private void FindVRHead() {

GameObject head = GameObject.FindWithTag("VRCamera");

if (head) {

vrHead = head.transform;

}

}

Listing 1. Szukanie obiektu kamery VR

W listingu 1 szukany jest obiekt z etykietą *VRCamera* i przypisywany do zmiennej. Podobne funkcje odpowiadają za szukanie lewego oraz prawego modelu dłoni.

Aby zmieniać pozycję dłoni wymagane było przepisanie wartości z pozycji oraz rotacji dłoni w świecie. Należy jednak zauważyć, że z uwagi na to, jak skonstruowany został model postaci potrzebna była dodatkowa rotacja. Kod odpowiadający za ruch znajduje się w listingu 2.

leftHand.transform.position = vrLeftWrist.position;

leftHand.transform.rotation = vrLeftWrist.rotation;

leftHand.transform.Rotate(90f, 0f, -90f);

rightHand.transform.position = vrRightWrist.position;

rightHand.transform.rotation = vrRightWrist.rotation;

rightHand.transform.Rotate(-90f, 180f, -90f);

Listing 2. Ustawianie i przekształcanie pozycji i transformacji obiektów

Rotacja w unity przedstawiana jest za pomocą zmiennych typu *Quaternion[[10]](#footnote-10),[[11]](#footnote-11)*, aby uprościć obliczenia rotacji dla komputera. Aby jednak ułatwić pracę programistom, kwaterniony są przekształcane na kąty Eulera. Przekształcanie to przydaje się w wielu momentach. Jednym z nich była chęć uniemożliwienia rotacji modelu w sytuacjach, które sprawiałyby, że model wyglądałby źle, np. model głowy skierowany byłby do środka ciała. Do ograniczenia rotacji posłużył kod poniżej, który blokuje za duże obroty głowy względem ciała na osiach X i Z.

transform.position = new Vector3(

vrHead.position.x,

transform.position.y,

vrHead.position.z);

if (head.transform.rotation.eulerAngles.x > 60f && head.transform.rotation.eulerAngles.x < 180f) {

head.transform.rotation = Quaternion.Euler(

60f,

head.transform.rotation.eulerAngles.y,

head.transform.rotation.eulerAngles.z);

}

if (head.transform.rotation.eulerAngles.z > 300f) {

head.transform.rotation = Quaternion.Euler(

head.transform.rotation.eulerAngles.x,

head.transform.rotation.eulerAngles.y,

300f);

} else if (head.transform.rotation.eulerAngles.z < 235f) {

head.transform.rotation = Quaternion.Euler(

head.transform.rotation.eulerAngles.x,

head.transform.rotation.eulerAngles.y,

235f);

}

Listing 3. Ustawianie pozycji głowy oraz ograniczanie nieprawidłowych rotacji

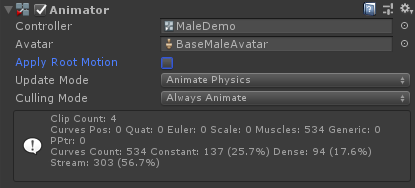
Listing 3 przedstawia kod używany do zmiany pozycji głowy. Położenie głowy na osi Y nie jest zmieniane. Skrypt generuje też nowe rotacje, jeśli te, które aktualnie są używane nie wyglądają poprawnie. Wartości które są nieodpowiednie zostały wyznaczone eksperymentalnie. Do utworzenia rotacji typu *Quaternion* na podstawie kątów Eulera służy funkcja *Quaterion.Euler().*

### 3.4.2. Odwrotna kinematyka postaci

Kinematyka odwrotna pozwala na zmianę pozycji elementów będącymi rodzicami przez obiekty będące dziećmi. Przykładem jest tutaj ruch dłoni, który zmienia układ rąk i tułowia.

W projekcie przyjęte zostało, że jeśli wartość absolutna różnicy rotacji kamery sterowanej kontrolerem na głowie na osi Y oraz aktualnej rotacji modelu postaci wynosi 30 lub więcej stopni to należy wykonać obrót tej postaci o 45 stopni w stronę obrotu kamery. Po obrocie wartość absolutna różnicy wynosi około 15 stopni.

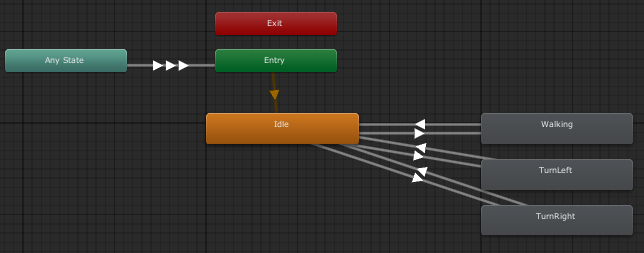
Aby obrót postaci był możliwy z animacjami w obiekcie postaci stworzony skonfigurowany został komponent *Animator*. Odpowiednie obiekty zostały pobrane i zaimportowane wraz z modelem. Konfiguracja przedstawiona jest na (Rys. 19).



Rysunek 23. Konfiguracja komponentu Animator, opracowanie własne

Następnie potrzebne było skonfigurowanie animacji. Okno *Animator* zostało uruchomione poprzez wybranie opcji *Window > Animation > Animator.* W karcie *Layers* przy *Base Layer* klikając ikonę zębatki włączono *IK Pass.* Na karcie *Parameters* zostały dodane trzy parametry: *AbsoluteMovement* typu *Float* oraz *TurnLeft* i *TurnRight* typu *Bool.*

Kolejnym krokiem było stworzenie stanów i przejść między nimi. Robi się to przez kliknięcie prawym klawiszem myszy w terminalu i wybranie opcji *Create State > Empty*. Cztery stany potrzebne do poruszania to *Idle, Walking, TurnLeft* i *TurnRight*. Idle jest stanem domyślnym i będzie on wykonywany zawsze, gdy . Zostało to zrobione przez wybranie opcji *Set as Layer Default State*. Poprzez opcję *Make Transition* między stanem *Idle* a *Walking, TurnLeft i TurnRight* zostało utworzona możliwość zmiany stanu.

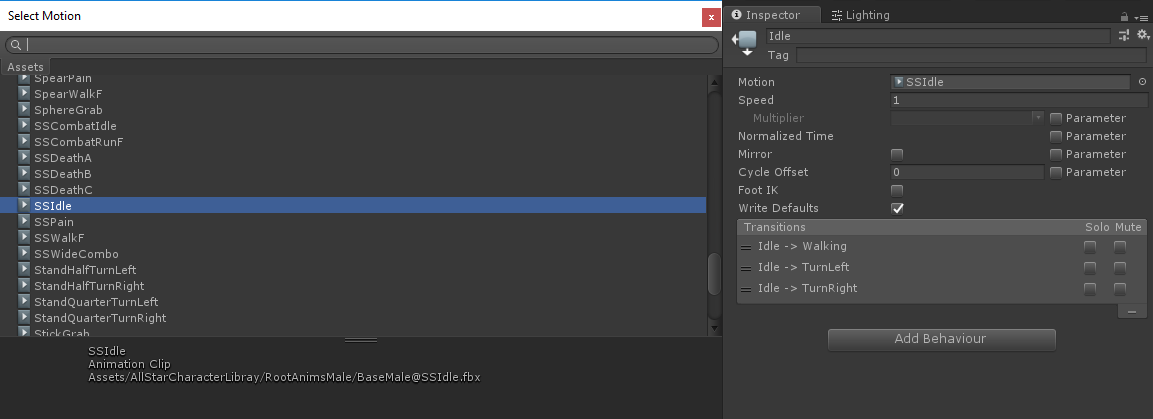


Rysunek 24. Wygląd maszyny stanów po dodaniu przejść między stanami, opracowanie własne

Przejście ze stanu *Idle* do *Walking* nie powinno następować przy niewielkich ruchach postaci, ale nie można też pozwolić, żeby użytkownik przemieszczał się całkowicie bez animacji. W sekcji *Conditions* wybrany został parametry *AbsoluteMovement Greater 0.03*. Natomiast przy przejściu ze stanu *Walking* do *Idle* ustawione zostały parametry *AbsoluteMovement Less 0.03.*Wartość 0.03 została wyznaczona doświadczalnie. W obu przejściach odznaczona została też opcja *Has Exit Time*. Opcja ta pozwala na przerwanie animacji jeśli warunek zmienił się przed jej ukończeniem.

W przypadku stanów zmiany ze stanu *Idle* do *TurnLeft* i *TurnRight* w sekcji *Conditions* ustawione zostały parametry *TurnLeft true* i *TurnRight true*. Przy przejściu w drugą stronę warunek został zmieniony na *false.* Tutaj także wyłączona została opcja *Has Exit Time*.

Kolejnym krokiem było skonfigurowanie animacji dla poszczególnych stanów. Pierwszym z nich był stan *Idle*. Dla tego stanu wybrana została animacja *SSIdle* pobrana wraz z modelem. W stanach *Walking, TurnLeft* i *TurnRight* zostały odpowiednio wybrane animacjeo nazwach *SSWalkF*, *StandQuarterTurnLeft* i *StandQuarterTurnRight.*



Rysunek 25. Wybór animacji dla stanu Idle, opracowanie własne

Po skonfigurowaniu animacji nastąpiła faza ich implementacji. Do obiektu postaci dodane zostały dwa puste obiekty, które zostały ustawione w miejscu, w którą stronę mniej więcej powinny być skierowane łokcie. Utworzony został nowy skrypt nazwie *IKHandling.* Skrypt ten podobnie jak *PlayerController* znajduje obiekty z etykietami *LeftWrist* i *RightWrist*. Na ich podstawie układane są animacje. Kod odpowiadający za zmianę modelu w zależności od położenia dłoni znajduje w listingu 4.

animator.SetIKPositionWeight(AvatarIKGoal.LeftHand, ikWeight);

animator.SetIKPosition(AvatarIKGoal.LeftHand, leftHandIKTarget.position);

animator.SetIKRotationWeight(AvatarIKGoal.LeftHand, ikWeight);

animator.SetIKRotation(AvatarIKGoal.LeftHand, leftHandIKTarget.rotation);

animator.SetIKHintPositionWeight(AvatarIKHint.LeftElbow, ikWeight);

animator.SetIKHintPosition(AvatarIKHint.LeftElbow, hintLeftHand.position);

Listing 4. Ustawianie opcji animatora

Skrypt ustawia pozycję, rotację i wskazanie miejsca skierowania łokci oraz wagę poszczególnych elementów. W projekcie wszystkie wagi zostały ustawione na 1. *IKHandling* sprawia, że pozycja rąk zmienia się w zależności od ruchu dłoni.

Kolejnym stworzonym elementem było uwzględnianie ruchów modelu przy ruchu głowy i obrotów w konkretną stronę. W *PlayerController* obliczana jest szybkość ruchu i ustawiany jest odpowiedni parametr animatora, co prezentuje listing 5.

float absoluteMovement = (Mathf.Abs(transform.position.x - vrHead.position.x) + Mathf.Abs(transform.position.z - vrHead.position.z)) \* Time.deltaTime \* 1000f;

animator.SetFloat("AbsoluteMovement", absoluteMovement);

Listing 5. Obliczanie wartości absolutnej ruchu postaci w przestrzeni

Następnie należało obliczyć różnicę kątów między rotacją głowy a rotacją postaci oraz jeśli rotacja ta jest większa niż 30 stopni to podjąć odpowiednią akcję, chyba że użytkownik jest w trakcie animacji. Podejmowane akcje to blokada animacji, zmiana kąta, który ma osiągnąć postać oraz uruchomienie odpowiedniej animacji. Jeśli użytkownik jest w trakcie animacji to powinna nastąpić rotacja postaci w tym kierunku. Implementację powyższych założeń przedstawia kod w listingu 6.

if (!lockRotation) {

float phi = Mathf.Abs(a - b) % 360f;

float distance = phi > 180f ? 360f - phi : phi;

float signedDistance = distance \* ((a - b >= 0 && a - b <= 180) || (a - b <= -180 && a - b >= -360) ? 1 : -1);

if (signedDistance > 30f) {

lockRotation = true;

StartCoroutine(RunFinishTurnTask());

neededRotation = Quaternion.Euler(0f, a - 45f, 0f);

animator.SetBool("TurnLeft", true);

} else if (signedDistance < -30f) {

lockRotation = true;

StartCoroutine(RunFinishTurnTask());

neededRotation = Quaternion.Euler(0f, a + 45f, 0f);

animator.SetBool("TurnRight", true);

}

} else {

transform.rotation = Quaternion.RotateTowards(transform.rotation, neededRotation, Time.deltaTime \* 180f);

}

Listing 6. Obliczanie kierunku obrotu i jego wykonanie



Rysunek 26. Wygląd postaci po zmianach w odwrotnej kinematyce, opracowanie własne

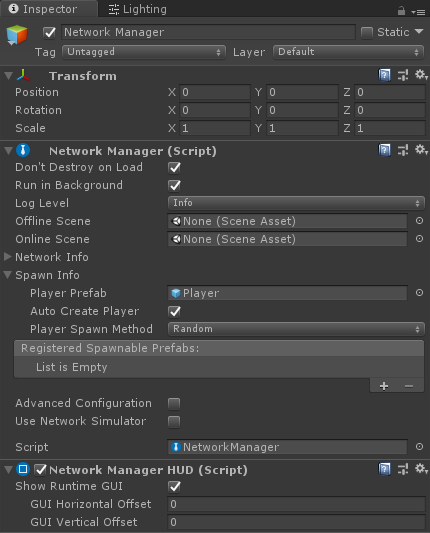
## 3.5. KOMUNIKACJA SIECIOWA

### 3.5.1. Wybór API

Przy wyborze API w projekcie najważniejszymi kryteriami była pełna obsługa synchronizacji wszystkich niezbędnych transformacji oraz niewielki nakład pracy. Z uwagi na chęć przesyłania dużej ilości informacji przy transformacjach kości dłoni i brak doświadczenia z Photon Unity Networking trudno było stwierdzić, czy 500 wiadomości na sekundę byłoby wystarczającą ilością. Z kolei Low Level API jest co prawda najbardziej elastycznym rozwiązaniem, ale ze względu na ogromną ilość pracy konieczną do wykonania rozwiązania, które byłoby wydajne, wybór padł na High Level API.

### 3.5.2. Synchronizacja ruchu postaci

Przed rozpoczęciem synchronizacji obiekt postaci należało zapisać jako prefab[[12]](#footnote-12). Aby synchronizacja przez Internet zadziałała utworzono nowy pusty obiekt, który został nazwany *Network Manager*. Do tego obiektu dodane zostały komponenty, skrypty *Network Manager* i *Network Manager HUD*. Pierwszy zapewnia komunikację sieciową, a drugi prosty interfejs pozwalający połączyć się z serwerem. W *Network Managerze* najważniejsze było ustawienie zmiennej *Player Prefab* na stworzony wcześniej prefab.



Rysunek 27. Konfiguracja Network Managera, opracowanie własne

Do stworzonego prefabu dodane teraz zostały komponenty *Network Identity* i *Network Transform*. W pierwszym z nich należy włączyć opcję *Local Player Authority*, a w drugimzmienić *Rotation Axis* na *XYZ (full 3D),* aby zmiany w ruchu obiektu były przekazywane niezależne od osi, na której występują.

W skryptach *PlayerController* i *IKHandling* wymagane są zmiany, ponieważ w obecnym stanie skrypty dla obiektów postaci będą się także uruchamiać na komputerach innych graczy. Należy sprawdzić, czy obiekt jest lokalnym obiektem gracza i tylko wtedy uruchomić.

Należy także zauważyć, że w tym momencie obiekty dłoni znajdowały się poza obiektem postaci. Nie jest jednak możliwe przeniesienie ich do tego obiektu, gdyż posiadają one skrypty *Steam VR\_Behaviour\_Skeleton*, które muszą być używane lokalnie, na komputerze użytkownika. Obiekty te zostały jednak skopiowane, a skrypt ten został usunięty. Brak tego skryptu nie pozwala jednak na ruchy dłoni. Dodatkowym skryptem, uruchamianym tylko lokalnie, który został do tego celu utworzony jest *HandMovementCopier*. Pozwala on skopiować dane transformacji wszystkich dzieci obiektu na inny obiekt o identycznej strukturze. Obiekty dłoni, do których zostaną przekazane dane transformacji ustalone są w prefabie, a obiekty dłoni kontrolerów ładowane są dynamicznie. Kopiowanie pozycji i rotacji następuje przy każdej ich zmianie i jest przedstawione w kodzie poniżej.

for (int i = 0; i < leftHandNodes.Count; ++i) {

leftHandNodes[i].position = vrLeftHandNodes[i].position;

leftHandNodes[i].rotation = vrLeftHandNodes[i].rotation;

}

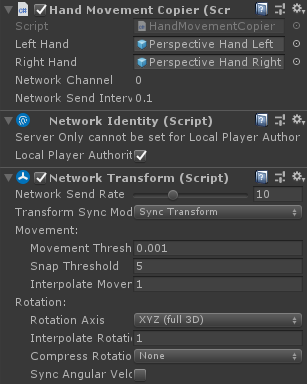
for (int i = 0; i < rightHandNodes.Count; ++i) {

rightHandNodes[i].position = vrRightHandNodes[i].position;

rightHandNodes[i].rotation = vrRightHandNodes[i].rotation;

}

Listing 7. Kopiowanie pozycji i rotacji dłoni sterowanej kontrolerem do kopii tej dłoni



Rysunek 28. Konfiguracja skryptów sieciowych obiektu postaci, opracowanie własne

### 3.5.3. Wyszukiwarka dostępnych serwerów

## 3.6. STAN GRY

## 3.7. DODAWANIE DŹWIĘKÓW

# TESTOWANIE

# ZAKOŃCZENIE

## 5.1. PODSUMOWANIE

## 5.2. MOŻLIWOŚCI ROZWOJU

# 



# BIBLIOGRAFIA

1. Unity User Manual (2018.2), <https://docs.unity3d.com/2018.2/Documentation/Manual/> (dostęp: 18.11.2018).
2. Oculus Documentation, <https://developer.oculus.com/documentation/> (dostęp: 13.12.2018).
3. OpenVR API Documentation, <https://github.com/ValveSoftware/openvr/wiki/API-Documentation> (dostęp: 18.11.2018).
4. Unity – Multiplayer Networking, <https://unity3d.com/learn/tutorials/topics/multiplayer-networking> (dostęp 18.11.2018)
5. ​Gańko Tomasz, Gra-zabawka dla niemowląt przygotowana z użyciem w Unity 3D, Toruń 2017
6. Ioannou Nikolaos, Multi-Player​ ​Virtual​ ​Reality​ ​Game based​ ​on​ ​Hand​ ​Tracking​ ​Interaction, Chania 2017.

# SPIS ILUSTRACJI

[Rysunek 1. Główne okno edytora Unity, opracowanie własne 6](#_Toc535178688)

[Rysunek 2. Schemat Oculus Rifta 7](#_Toc535178689)

[Rysunek 3. Kontrolery Oculus Touch 8](#_Toc535178690)

[Rysunek 4. Okno przypisywania przycisków do akcji, opracowanie własne 9](#_Toc535178691)

[**Rysunek 5. Okno główne Visual Studio 2018, opracowanie własne** 10](#_Toc535178692)

[Rysunek 6. Schemat działania Git, opracowanie własne 11](#_Toc535178693)

[Rysunek 7. Schemat warstwy sieciowej w Unity, opracowanie własne 12](#_Toc535178694)

[Rysunek 8. Sześć stopni swobody 14](#_Toc535178695)

[Rysunek 9. Przykład śledzenia pozycji dla VR 15](#_Toc535178696)

[Rysunek 11. Ekran tworzenia projektu w Unity, opracowanie własne 18](#_Toc535178697)

[Rysunek 12. Okno konfiguracji opcji pluginu SteamVR, opracowanie własne 19](#_Toc535178698)

[Rysunek 13. Konfiguracji Opcji XR, opracowanie własne 20](#_Toc535178699)

[Rysunek 14. Okno edytora z uruchomioną podstawową sceną, opracowanie własne 21](#_Toc535178700)

[Rysunek 15. Resetowanie komponentu w Edytorze, opracowanie własne 21](#_Toc535178701)

[Rysunek 16. Usuwanie komponentu Box Collider, opracowanie własne 22](#_Toc535178702)

[Rysunek 17. Tworzenie nowego materiału, opracowanie własne 23](#_Toc535178703)

[Rysunek 18. Wygląd utworzonej mapy wraz z pokazaną hierarchią obiektów, opracowanie własne 24](#_Toc535178704)

[Rysunek 19. Konfiguracja komponentu szkieletu dla lewej dłoni, opracowanie własne 25](#_Toc535178705)

[Rysunek 20. Efekty wyboru opcji Range of Motion: Without Controller (po lewej) i With Controller (po prawej), opracowanie własne 26](#_Toc535178706)

[Rysunek 21. Nieprawidłowy (po lewej) i prawidłowy (po prawej) układ kości dłoni w modelu, opracowanie własne 26](#_Toc535178707)

[Rysunek 22. Dodawanie etykiet do projektu, opracowanie własne 27](#_Toc535178708)

[Rysunek 23. Konfiguracja komponentu Animator, opracowanie własne 30](#_Toc535178709)

[Rysunek 24. Wygląd maszyny stanów po dodaniu przejść między stanami, opracowanie własne 30](#_Toc535178710)

[Rysunek 25. Wybór animacji dla stanu Idle, opracowanie własne 31](#_Toc535178711)

[Rysunek 26. Wygląd postaci po zmianach w odwrotnej kinematyce, opracowanie własne 33](#_Toc535178712)

[Rysunek 27. Konfiguracja Network Managera, opracowanie własne 34](#_Toc535178713)

[Rysunek 28. Konfiguracja skryptów sieciowych obiektu postaci, opracowanie własne 36](#_Toc535178714)

# DODATEK A. MODELE UŻYTE W PROJEKCIE

1. *Unity Asset Store, https://assetstore.unity.com/* [↑](#footnote-ref-1)
2. Schemat pochodzi z *Oculus Rift Development Kit 2 Quick Start Guide* [↑](#footnote-ref-2)
3. Immersja – proces zanurzania się użytkownika w rzeczywistość elektroniczną [↑](#footnote-ref-3)
4. Schemat pochodzi z *Oculus Touch, la manette VR, sera disponible le 6 décembre – Geeko*, <https://geeko.lesoir.be/2016/10/07/oculus-touch-la-manette-vr-sera-disponible-le-6-decembre/> (dostęp 09.01.2019) [↑](#footnote-ref-4)
5. Asset jest reprezentacją dowolnej rzeczy np. modelu, dźwięku, skryptu lub zestawu skryptów w projekcie. *Unity – Manual: Asset Workflow*, <https://docs.unity3d.com/Manual/AssetWorkflow.html>(dostęp 12.01.2019) [↑](#footnote-ref-5)
6. Paralaksa to efekt, który sprawia, że obiekty znajdujące się daleko od obserwującego będącego w ruchu wydają się przesuwać wolniej od tych blisko obserwującego, [↑](#footnote-ref-6)
7. *Six degrees of freedom – Wikipedia*, https://en.wikipedia.org/wiki/Six\_degrees\_of\_freedom (dostęp: 12.01.2019) [↑](#footnote-ref-7)
8. *How does Oculus Quest work? – Quora*, https://www.quora.com/How-does-the-Oculus-Quest-work (dostęp 12.01.2019) [↑](#footnote-ref-8)
9. Sci Fi Officer Captain w Unity Asset Store, https://assetstore.unity.com/packages/3d/characters/sci-fi-officer-captain-52376 (dostęp: 19.11.2018) [↑](#footnote-ref-9)
10. Kwaterniony – struktura algebraiczna będąca rozszerzeniem ciała liczb zespolonych, https://pl.wikipedia.org/wiki/Kwaterniony (dostęp 10.01.2019) [↑](#footnote-ref-10)
11. *Unity – Scripting API: Quaternion*, https://docs.unity3d.com/ScriptReference/Quaternion.html (dostęp: 10.01.2019) [↑](#footnote-ref-11)
12. Prefab jest obiektem zapisanym w plikach projektu i który działa jako swego rodzaju szablon dla innych obiektów. Kopie prefabów można dynamicznie tworzyć w aplikacji, *Unity – Manual: Prefabs,* <https://docs.unity3d.com/Manual/Prefabs.html> (dostęp 12.01.2019) [↑](#footnote-ref-12)